Scalable, Fast Cloud Computing with Execution Templates

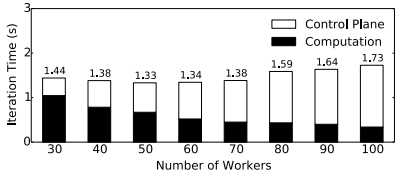
Omid Mashayekhi, Hang Qu, Chinmayee Shah, Philip Levis

Introduction and Motivation

• Available cloud frameworks either support fine-grained task scheduling or high task throughput, but not both.

➢ Systems such as Naiad and TensorFlow install static data flow graph for efficiency but sacrifice scheduling flexibility.

➢ Systems such as Spark schedule at the task granularity but only handle few thousands tasks per second.



Logistic regression in Spark 2.0 MLlib: Increasingly parallelizing reduces computation time (black bars) but control overhead outstrip these gains, *increasing* completion time.

• **Execution templates** introduce a new design point:

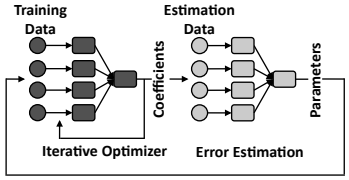
➢ For reoccurring computations cache the control decisions on computing nodes as templates and instantiate the templates with new parameters.

➢ Changes in scheduling are supported as edits in the installed templates. The cost of scheduling is proportional to the size of changes.

Execution Templates

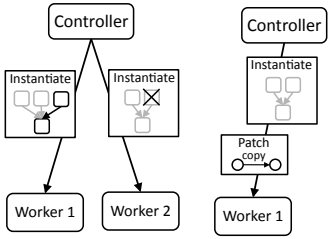
• **Basic Blocks**: execution templates cache control plane decisions at the granularity of basic blocks in the driver program. Unlike batching, execution templates are capable of handling nested-loops and data-dependent branches.

```
while (error > threshold_e) {
  while (gradient > threshold_g) {
    // Optimization code block
    gradient = Gradient(tdata, coeff, param
    coeff += gradient
  }
  // Estimation code block
  error = Estimate(edata, coeff, param)
  param = update_model(param, error)
}
```



• **Edit**: minor changes in the scheduling, for example task migrations, reflect in the templates as in place edits added by controller upon instantiation.

• **Patch**: templates are not bound to a static control flow. Controller can patch the worker state to enforce the required preconditions of the templates.



Edits: migrating task from worker 2 to worker 1.

Patching the worker state.

Implementation

• We have implemented execution templates in a cloud computing framework called **Nimbus**.

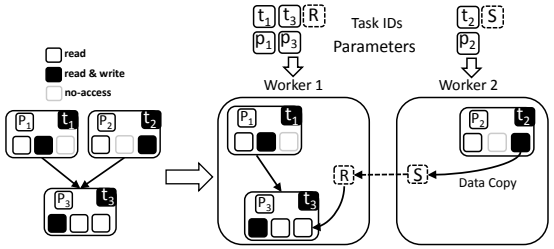
• Execution templates cache the control dependency between tasks, data access patterns, and task executables.

• Workers can queue tasks and resolve dependencies locally.

• Inter-worker dependencies are encoded as data copy commands; workers exchange data directly.

• Nimbus has a mutable data model which allows caching the data access patterns within the template.

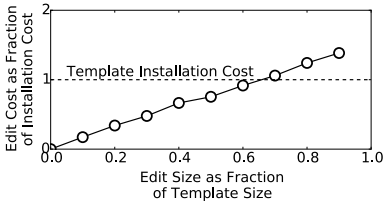
• Templates are instantiated by passing new task identifiers and parameters to the workers.



Evaluation

Fine-Grained Scheduling

•The cost of edits is proportional to the size of scheduling changes (single edit costs 41µs).

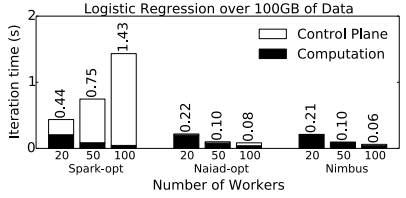


Task Throughput

• Although Nimbus has a centralized controller similar to Spark, it handles orders of magnitude higher task throughput.

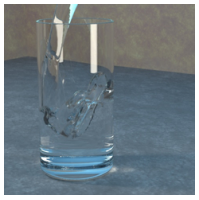
Data Analytics

• Nimbus with execution templates matches the performance of distributed frameworks with static data flow (Naiad) while keeping the scheduling granularity (Spark).



HPC Applications

• Execution templates allow running complex water simulation (**PhysBAM**) with triply-nested loop and data dependent branches within 15% of MPI performance.



#cell (#parts)	entire step		solver iteration	
	#access	rate	#access	rate
512^3 (64)	1.5M	63Kps	13K	92Kps
1024^3 (512)	12.5M	394Kps	102K	463Kps